
aioswitcher

Release 2019.10.25

Tomer Figenblat

Mar 27, 2020

CONTENTS

1	What	1
2	Install	3
3	Prerequisites	5
4	Usage	7
5	License	17
6	Change Log	19
7	Credits	21
8	Projects using this module	23
9	Code of Conduct	25
10	Contributing	27
11	Code Documentation	33
	Python Module Index	55
	Index	57

WHAT

PyPi module named `aioswitcher` for integrating with the `Switcher Water Heater`.

This module:

- Is packaged with `poetry`.
- Works concurrently using Python's Asynchronous I/O module `asyncio`.
- Is static typed and checked with `mypy` based on `PEP484`.
- Follows `black` code style rules and guidelines.
- Package is described using `pyproject.toml` based on `PEP517` and `PEP518`.

Although the `aioswitcher` module requires the use of *Python 3.5/3.6/3.7*, The use of *Python 3.7* is preferable.

INSTALL

```
pip install aioswitcher
```

That's it!

Note:

- `Switcher V2` was verified by the author of this project.
 - `Switcher Touch` has been verified by various users of this project.
-

PREREQUISITES

- Install and configure your Switcher device.
- Collect the following information from the device's following NightRang3r instructions in the [Switcher-V2-Python](#) repository:
 - ip_address
 - phone_id
 - device_id
 - device_pass

USAGE

This module provides two separate integrations with the Switcher device, both fully asynchronous.

The first integration is a *UDP Bridge* constantly listening to broadcast messages launched from the device approximately every 4 seconds. This integration provides constant device state updates.

The second integration is a *TCP Socket API* providing the ability, to not only get the current state of the device, but also control it.

TOC

- *UDP Bridge*
 - *Example of UDP Bridge usage*
- *TCP Socket API*
 - *Example of TCP Socket API usage*
- *Objects and Properties*
 - *SwitcherV2Device*
 - *SwitcherV2Schedule*
- *API Response Messages*
 - *SwitcherV2BaseResponseMSG*
 - *SwitcherV2StateResponseMSG*
 - *SwitcherV2ControlResponseMSG*
 - *SwitcherV2SetAutoOffResponseMSG*
 - *SwitcherV2UpdateNameResponseMSG*
 - *SwitcherV2GetScheduleResponseMSG*
 - *SwitcherV2DisableEnableScheduleResponseMSG*
 - *SwitcherV2DeleteScheduleResponseMSG*
 - *SwitcherV2CreateScheduleResponseMSG*

4.1 UDP Bridge

With the *UDP Bridge* integration, we're constantly listening to messages broadcast from the device containing information about the device and the device's state.

The message is being broadcast from the device approximately every 4 seconds, once the message is received and verified it will be put in a `asyncio.Queue` with the max size of 1, which means the `Queue` always has 1 or `None` messages waiting, containing the latest state received.

Each message will be an updated instance of the *SwitcherV2Device* object.

The bridge can be used as a `Context Manager` as well as being instantiated and controlled manually.

Note: The *UDP Bridge* integration will allow you to receive **Real-Time** updates from the device approximately every 4 seconds, yet it will not allow you to control the device.

For controlling the device you can use the *TCP Socket API* integration. (Both work simultaneously).

4.1.1 Example of UDP Bridge usage

```
import asyncio
from datetime import datetime
from aioswitcher.bridge import SwitcherV2Bridge
from aioswitcher.devices import SwitcherV2Device

# instructions on getting this data is in,
# https://github.com/NightRang3r/Switcher-V2-Python
phone_id = "your_devices's_phone_id"
device_id = "your_devices's_device_id"
device_password = "your_devices's_device_password"

# create a new event loop
your_loop = asyncio.get_event_loop()

"""Use as instance."""
async def run_as_instance() -> None:
    v2bridge = SwitcherV2Bridge(
        your_loop, phone_id, device_id, device_password
    )
    # start the bridge
    await v2bridge.start()

    # get the Queue
    queue = v2bridge.queue # type: asyncio.Queue

    # create an event to signal your coroutine to start/stop
    signal_event = asyncio.Event()
    signal_event.set()

    # coroutine for getting the device from the queue
    async def get_device_from_queue() -> None:
        device = await queue.get() # type: SwitcherV2Device
        print("instance state is: {}".format(device.state))
        print(datetime.now())
        if signal_event.is_set():
```

(continues on next page)

(continued from previous page)

```

        your_loop.call_soon(get_device_from_queue)
    return None

    # schedule your coroutine which will wait for the device
    # to be put in the queue, and print the time and its state
    # afterwards it will call itself again, the result will be
    # the device state being printed every approximately 4 seconds
    your_loop.call_soon(get_device_from_queue)

    # wait for 40 seconds
    # the state should be printed about 8 to 10 times
    await asyncio.sleep(40)

    # stop the coroutine
    signal_event.clear()

    # stop the bridge
    await v2bridge.stop()

    return None

"""Use as context manager."""
async def run_as_context_manager() -> None:
    async with SwitcherV2Bridge(
        your_loop,
        phone_id,
        device_id,
        device_password,
    ) as v2bridge:
        # get the Queue
        queue = v2bridge.queue # type: asyncio.Queue

        # create an event to signal your coroutine to start/stop
        signal_event = asyncio.Event()
        signal_event.set()

        # coroutine for getting the device from the queue
        async def get_device_from_queue() -> None:
            device = await queue.get() # type: SwitcherV2Device
            print("context manager state is: {}".format(device.state))
            print(datetime.now())
            if signal_event.is_set():
                your_loop.call_soon(get_device_from_queue)
            return None

        # schedule your coroutine which will wait for the device
        # to be put in the queue, and print the time and its state
        # afterwards it will call itself again, the result will be
        # the device state being printed every approximately 4 seconds
        your_loop.call_soon(get_device_from_queue)

        # wait for 40 seconds
        # the state should be printed about 8 to 10 times
        await asyncio.sleep(40)

        # stop the coroutine
        signal_event.clear()

```

(continues on next page)

(continued from previous page)

```

    return None

your_loop.run_until_complete(run_as_instance())
your_loop.run_until_complete(run_as_context_manager())

loop.close()

```

4.2 TCP Socket API

With *TCP Socket API* integration we gain the following abilities:

- Get the device status
- Control the device
- Get the schedules from the device
- Set the device name
- Set the device Auto-Off configuration
- Create/Delete/Enable/Disable schedules on the device.

Note: Although the *TCP Socket API* is applicable as a context manager and as an instance of an object, It is preferable to use it as a context manager due to the nature of the tcp connection (you don't want to occupy a connection slot on the device any longer then you have to or you'll start seeing `TimeOutErrors`).

To use as an instance (which will not be covered here), you can rely on the `UDP Bridge` example and just substitute `start()` and `stop()` with `connect()` and `disconnect()`.

The various responses are covered in the *API Response Messages* section.

4.2.1 Example of TCP Socket API usage

```

import asyncio
from datetime import timedelta
from aioswitcher import consts, tools
from aioswitcher.api import SwitcherV2Api, messages
from aioswitcher.schedules import SwitcherV2Schedule

# create a new event loop
your_loop = asyncio.get_event_loop()

# if you're also using the udp bridge,
# the ip address is available at (SwitcherV2Device).ip_addr
ip_address = "your_device's_ip_address"

# instructions on getting this data is in
# https://github.com/NightRang3r/Switcher-V2-Python
phone_id = "your_devices's_phone_id"
device_id = "your_devices's_device_id"
device_password = "your_devices's_device_password"

"""Use as context manager."""
async def run_as_context_manager() -> None:

```

(continues on next page)

(continued from previous page)

```

async with SwitcherV2Api(
    your_loop, ip_address, phone_id,
    device_id, device_password) as swapi:
    # get the device state
    # response: messages.SwitcherV2StateResponseMSG
    state_response = await swapi.get_state()

    # control the device: on / off / on + (15/30/45/60) minutes timer
    # response: messages.SwitcherV2ControlResponseMSG
    turn_on_response = await swapi.control_device(
        consts.COMMAND_ON)
    turn_off_response = await swapi.control_device(
        consts.COMMAND_OFF)
    turn_on_30_min_response = await swapi.control_device(
        consts.COMMAND_ON, '30')

    # set the limit time to auto-shutdown the device (1 < hours < 24)
    # response: messages.SwitcherV2SetAutoOffResponseMSG
    time_to_off = timedelta(hours=1, minutes=30)
    set_auto_off_response = await swapi.set_auto_shutdown(time_to_off)

    # set the device name (2 < length < 33)
    # response: messages.SwitcherV2UpdateNameResponseMSG
    set_name_response = await swapi.set_device_name("new device name")

    # get the configured schedules from the device
    # response: messages.SwitcherV2GetScheduleResponseMSG
    get_schedules_response = await swapi.get_schedules()

    # disable or enable a schedule
    # schedule_data = (SwitcherV2Schedule).schedule_data
    # response: messages.SwitcherV2DisableEnableScheduleResponseMSG
    #
    # the following will enable the schedule:
    # updated_schedule_data = (
    #     schedule_data[0:2] + consts.ENABLE_SCHEDULE + schedule_data[4:])
    #
    # the following will disable the schedule:
    # updated_schedule_data = (
    #     schedule_data[0:2] + consts.DISABLE_SCHEDULE + schedule_data[4:])
    enable_disable_response = await swapi.disable_enable_schedule(
        updated_schedule_data)

    # delete a schedule (0 <= schedule_id <= 7)
    # schedule_id = (SwitcherV2Schedule).schedule_id
    # response: messages.SwitcherV2DeleteScheduleResponseMSG
    delete_response = await swapi.delete_schedule(schedule_id)

    # create a schedule to turn on at 20:30 and off at 21:00
    # response: messages.SwitcherV2CreateScheduleResponseMSG
    schedule_days = [0]
    # append selected days, if non-recurring skip next
    schedule_days.append(consts.DAY_TO_INT_DICT[consts.SUNDAY])
    schedule_days.append(consts.DAY_TO_INT_DICT[consts.MONDAY])
    schedule_days.append(consts.DAY_TO_INT_DICT[consts.TUESDAY])
    schedule_days.append(consts.DAY_TO_INT_DICT[consts.WEDNESDAY])
    schedule_days.append(consts.DAY_TO_INT_DICT[consts.THURSDAY])

```

(continues on next page)

(continued from previous page)

```
schedule_days.append(consts.DAY_TO_INT_DICT[consts.FRIDAY])
schedule_days.append(consts.DAY_TO_INT_DICT[consts.SATURDAY])
# skip here if non-recurring
weekdays = await tools.create_weekdays_value(
    your_loop, schedule_days)
start_time = await tools.timedelta_str_to_schedule_time(
    your_loop, str(timedelta(hours=20, minutes=30)))
end_time = await tools.timedelta_str_to_schedule_time(
    your_loop, str(timedelta(hours=21)))
schedule_data = consts.SCHEDULE_CREATE_DATA_FORMAT.format(
    weekdays, start_time, end_time)
create_response = await swapi.create_schedule(schedule_data)

return None

your_loop.run_until_complete(run_as_context_manager())

your_loop.close()
```

4.3 Objects and Properties

There are two main objects you need to be aware of:

- The first object is the one representing the device itself, `aioswitcher.devices.SwitcherV2Device` *SwitcherV2Device*.
- The second object is the one representing the device's schedule, `aioswitcher.schedules.SwitcherV2Schedule` *SwitcherV2Schedule*.

4.3.1 SwitcherV2Device

Property	Type	Description	Possible Values	Default Value
device_id	str	Return the device id.	ab1c2d	
ip_addr	str	Return the ip address.	192.168.100.157	wait- ing_for_data
mac_addr	str	Return the mac address.	A1:B2:C3:45:67:D8	wait- ing_for_data
name	str	Return the device name.	device name	wait- ing_for_data
state	str	Return the device state.	on, off	
remaining_time	str	Return the auto-off configuration value.	%H:%M:%S	wait- ing_for_data
auto_off_set	str	Return the time left to auto-off.	%H:%M:%S	wait- ing_for_data
power_consumption	int	Return the power consumption in watts.	2780	0
electric_current	float	Return the power consumption in amps.	12.8	0.0
phone_id	str	Return the the phone id.	1234	
last_data_update	datetime	Return the timestamp of the last update.	%Y-%m- %dTH:%M:%S.%F	
last_state_change	datetime	Return the timestamp of the last state change.	%Y-%m- %dTH:%M:%S.%F	

4.3.2 SwitcherV2Schedule

Property	Type	Description	Possible Values	Default
schedule_id	str	Return the schedule id.	0-7	
enabled	bool	Return true if enabled. Has a setter manipulating the schedule status.	True, False	False
recurring	bool	Return true if recurring.	True, False	False
days	List[str]	Return the weekdays of the schedule.	<ul style="list-style-type: none"> • Sunday • Monday • Tuesday • Wednesday • Thursday • Friday • Saturday • Every day 	
start_time	str	Return the start time of the schedule.	%H:%M	waiting_for_data
end_time	str	Return the end time of the schedule.	%H:%M	waiting_for_data
duration	str	Return the duration time of the schedule.	0:30:00	waiting_for_data
schedule_data	str	Return the schedule data for managing the schedule. has a setter manipulating the schedule data.	Any	waiting_for_data
init_future	asyncio.Future	Return the future of the init.	SwitcherV2Schedule	

4.4 API Response Messages

The following are the response message objects returning from the various API functions. The source of the responses can be found `aioswitcher.api.messages`.

Please note the `aioswitcher.api.messages.ResponseMessageType Enum Class` for identifying the response message types:

- *AUTO_OFF*
- *CONTROL*
- *CREATE_SCHEDULE*
- *DELETE_SCHEDULE*
- *DISABLE_ENABLE_SCHEDULE*
- *GET_SCHEDULES*

- *STATE*
- *UPDATE_NAME*

4.4.1 SwitcherV2BaseResponseMSG

Property	Type	Description
unparsed_response	bytes	Return the unparsed response message.
successful	bool	Return the status of the message.
msg_type	ResponseMessageType	Return the response message type.

4.4.2 SwitcherV2StateResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type ResponseMessageType.STATE

Property	Type	Description
state	str	Return the state. Possible values are: <ul style="list-style-type: none"> • aioswitcher.consts.STATE_ON • aioswitcher.consts.STATE_OFF
time_left	str	Return the time left to auto-off.
auto_off	str	Return the auto-off configuration value.
power	Optional[int]	Return the current power consumption in watts.
current	Optional[float]	Return the power consumption in amps.
init_future	asyncio.Future	Return the future of the initialization. As the initialization of this message requires some asynchronous actions, please use <code>init_future.result()</code> to get the message object.

4.4.3 SwitcherV2ControlResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type ResponseMessageType.CONTROL

No properties are added by `object`.

4.4.4 SwitcherV2SetAutoOffResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.AUTO_OFF`

No properties are added by `object`.

4.4.5 SwitcherV2UpdateNameResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.UPDATE_NAME`

No properties are added by `object`.

4.4.6 SwitcherV2GetScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.GET_SCHEDULES`

Property	Type	Description
found_schedules	<code>bool</code>	Return true if found schedules in the response.
get_schedules	<code>List[SwitcherV2Schedule]</code>	Return a list of <i>SwitcherV2Schedule</i> .

4.4.7 SwitcherV2DisableEnableScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.DISABLE_ENABLE_SCHEDULE`

No properties are added by `object`.

4.4.8 SwitcherV2DeleteScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.DELETE_SCHEDULE`

No properties are added by `object`.

4.4.9 SwitcherV2CreateScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.CREATE_SCHEDULE`

No properties are added by `object`.

LICENSE

MIT License

Copyright © 2019 Tomer Figenblat

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHANGE LOG

6.1 2019.10.25

2019.10.25 Full Changelog

- Dependencies and requirements version updates.

6.2 2019.8.28

2019.8.28 Full Changelog

- Removed transitive dependencies from requirements file.
- Update dependencies and requirements.
- Incorporated .editorconfig file.
- RestructuredText are now LF and not CRLF.
- Cleanups.

6.3 2019.7.2

2019.7.2 Full Changelog

- Update dependencies and requirements.
- Added public templates.
- Removed TravisCI.
- Removed Coveralls.
- Better tox tests.
- Added NPM tools for testing.
- **Migrated to Poetry.**
- **Integrated pyproject.toml.**
- **Migrated to black code formatter.**
- Added documentation site including **code documentation.**

6.4 2019.6.11

2019.6.11 Full Changelog

- Added security testing with Bandit, Codacy and CircleCi.
- Added continues delivery to CircleCi's workflow.
- Various fix-ups and clean-ups.
- Updated various dependencies.

6.5 2019.6.11

2019.6.11 Full Changelog

- Added security testing with Bandit, Codacy and CircleCi.
- Added continues delivery to CircleCi's workflow.
- Various fix-ups and clean-ups.
- Updated various dependencies.

6.6 2019.4.26

2019.4.26 Full Changelog

- Unit-tests
- Continuous integration
- Code coverage

CREDITS

This project wouldn't have been able to happen without the amazing work performed by NightRang3r and AviadGolan in the [Switcher-V2-Python](#) project.

So... Thanks!

PROJECTS USING THIS MODULE

- Home Assistant's `switcher_kis` component:
 - [switcher_kis code](#)
 - [switcher_kis documentation](#)
- Switcher Boiler unofficial docker-based WebAPI:
 - [webapi code](#)
 - [webapi documentation](#)

Note: If you use this module in your open-source project and you want it to be listed here, please let me know (or add it yourself via github).

CODE OF CONDUCT

9.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socioeconomic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

9.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

9.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

9.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

9.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at tomer.figenblat@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

9.6 Attribution

This Code of Conduct is adapted from the [Contributor Covenant](#), version 1.4, available [here](#).

For answers to common questions about this code of conduct, see [faq](#).

CONTRIBUTING

First off, thank you for taking the time to contribute.

Contributing is pretty straight-forward: * Fork the repository * Commit your changes * Create a pull request against the **dev** branch

Please feel free to contribute, even to this contributing guideline file, if you see fit.

TOC

- *Items description*
 - *Configuration files*
 - *Python*
 - *Requirement files*
 - *Package management*
 - *Documentation*
- *Continuous Integration*
 - *CircleCi*
 - *CodeCov*
 - *Codacy*
 - *Requires-io*
 - *David-DM*
- *Continuous Deployment*
 - *Read the Docs*
 - *PyPI*
- *Environments and Tools*
- *Testing*
- *Guidelines*
 - *NPM Scripts*
- *Code of Conduct*

10.1 Items description

10.1.1 Configuration files

- `.circle/config.yml` is the configuration file for [CircleCi Continuous Integration and Deployment Services](#).
- `.codecov.yml` is the configuration file for [CodeCov Code Coverage](#).
- `.coveragerc` is the configuration file for [Coverage.py](#) creating coverage reports with the `pytest-cov` plugin.
- `.yamllint` is the configuration for [yamllint A Linter for YAML Files](#) linting `yml` files.
- `.remarkrc` is the configuration file for [remark-lint](#) plugin for [Remark](#) linting `markdown` files.
- `bandit.yml` is the configuration file for [Bandit common security issues finder](#) checking python scripts.
- `.spelling` is the dictionary file used by both [markdown-spellcheck](#) and [scspell3k](#). Case-insensitive words in this file will not raise a spelling mistake error.

10.1.2 Python

- `src/aioswitcher` is the *Python* modules making the package.
- `tests` is where *Python* test-cases are stored and executed with `pytest`.
- `pyscripts`` is where *Python* scripts are stored.

10.1.3 Requirement files

- `poetry.lock` is the lock file describing the module version tree of the `pypi` modules. This helps locking down working versions of modules, it is part of the [poetry](#) dependency management.
- `package-lock.json` is the lock file describing the module version tree of the `npm` modules. This helps locking down working versions of modules, it is part of the [npm](#) dependency management.
- `requirements.txt` has nothing to do directly with the package structure. We use [poetry](#) for packaging and building. This file is actually being manually build with `pyscripts/poetry-to-requirements.py` from the content of `poetry.lock` file for legacy support (e.g. [Requires-io](#)).

10.1.4 Package management

- The `package.json` file specified by [npm](#) manages our dependencies, scripts and some metadata.
- The `pyproject.toml` is the main configuration file for the `pypi` package based on [PEP518](#). Please note, this package is being managed, build, packaged and deployed with [poetry](#).

10.1.5 Documentation

- `docs/sources` is where the *restructuredText* files for creating the [Sphinx Documentation](#). are stored for build, deployment and hosting by [Read the Docs](#).
- `docs/Makefile` the basic *Makefile* for [Sphinx](#) documentation generator. From the `docs` path, type `make html` and `sphinx` will create the documentation site locally in `docs/build`.

10.2 Continuous Integration

10.2.1 CircleCi

By hook configuration, for every pull request, **CircleCi** will execute the workflows described in `.circleci/config.yml` and update the PR conversation with the results.

As a final step, **CircleCi** will push the **Coverage.py XML Report** to both **CodeCov** for code coverage analysis and **Codacy** for code quality analysis.

Both will of course push their results into the PR conversation.

Please note, **Codacy** is actually getting notified for the PR by a **GitHub** hook. The report being uploaded is for the dashboard presentation and does not trigger further action.

Some of the steps are considered required and may prevent the PR from being merged. But no worries, everything is fixable.

10.2.2 CodeCov

CodeCov is keeping tabs on our code coverage. When a report is uploaded (by **CircleCi**), **CodeCov** will check our code coverage and push its conclusions to PR conversation.

10.2.3 Codacy

Codacy is here to check the quality of our code. When a PR is created or updated, **GitHub** is hooked to notify **Codacy** that starts checking the quality of our code and push its conclusions to the PR conversation.

10.2.4 Requires-io

Requires.io is keeping an eye for versions updates upon the *Python* requirements listed in our legacy `requirements.txt` file.

10.2.5 David-DM

David-DM is keeping an eye for versions updates upon the *Npm* requirements listed in the `package.json` file.

10.3 Continuous Deployment

10.3.1 Read the Docs

By hook configuration, **Read the Docs** will build the documentation site based on `docs/source` and host it:

- `stable` tag [here](<https://aioswitcher.readthedocs.io/en/stable/>) will be built for every release snapshot.
- `latest` tag [here](<https://aioswitcher.readthedocs.io/en/latest/>) will be built for every push the dev branch, so it'll reflect unreleased changes.

10.3.2 PyPI

As for now, I'm not auto-deploying anything to [PyPI](#). Packages are being deployed manually.

10.4 Environments and Tools

Note: Python, poetry and Tox needs to be pre-installed.

- [Python](#), CPython interpreter based, although this package supports *Python3.5/3.6/3.7*, *Python3.7* is preferred.
- [Poetry](#) is being used for packaging and dependency management. * Please install [Poetry](#) if you plan on developing or testing the package.
- [Tox](#) for automating unit testing in your local environment. * Please install [Tox](#) if you want to perform local testing automation.
 - Tox utilizes Python's [virtualenv](#).
 - Tox is configured with `pyproject.toml`.
 - To run tox, simply execute `tox` from the `pyproject.toml`'s path. It is recommended that you also run `tox --help` to get familiar with the various options such as `-e` and `-r` that will help you perform faster and better tests.

Note: The rest of the steps require no installation on your behalf, but knowing them is important seeing they are key elements for testing with [Tox](#) and/or [CircleCi](#).

- *Python Module:* [nodeenv](#), a tool that enables us to create a Node.js virtual environment in resemblance to [virtualenv](#), this tool also allows combining [nodeenv](#) within [virtualenv](#), which is exactly what we're doing with `tox`.
- *NPM Package:* [package-json-validator](#) for validating the `package.json` file.
- *Python Package:* [yamllint](#) for linting the project `yml` files. * [yamllint](#) is configured with `.yamllint`.
- *NPM Package:* [markdown-spellcheck](#) for checking the project `markdown` files for spelling errors.
 - [markdown-spellcheck](#) dictionary file is `.spelling`.
- *NPM Package:* [remark-lint](#) which is a plugin for [remark](#) and the `remark-cli` command line tool for linting `markdown` files residing at the `base path` and in `.github`.
 - [remark-lint](#) uses a couple of presets and tools, all can be found under the `dependencies` key in `package.json`.
 - [remark-lint](#) is configured with `.remarkrc`.
- *Python Module:* [doc8](#) for checking `restructuredText` syntax for files residing in `docs/source` used to create the documentation site.
- *Python Module:* [scspell3k](#) for spell checking `restructuredText` files residing in `docs/source` used to create the documentation site. * [scspell3k](#) dictionary file is `.spelling`.
- *Python Module:* [sphinx](#) for building the documentation site from the `restructuredText` files residing in `docs/source`.
 - It's worth mentioning that the [documentation site](#), hosted with [Read the Docs](#) is based upon the theme `sphinx-rtd-theme`.

- *Python Package:* `bandit` for finding common security issues with against the *Python* files. * `bandit` is configured with `bandit.yml`.
- *Python Package:* `isort` for sorting *Python* imports. - `isort` is configured with `pyproject.toml`.
- *Python Package:* `flake8` for linting *Python* files.
- *Python Package:* `black` for formatting *Python* files. * `black` is configured with `pyproject.toml`.
- *Python Package:* `mypy` for checking static typing in *Python* files.
- *Python Package:* `pytest` as testing framework for running test-cases written in `tests`.

10.5 Testing

Testing is performed with `Pytest` Full-featured *Python* testing tool. The various test-cases is in `tests`.

For automated local tests, use `Tox`.

10.6 Guidelines

Note: The project `semver` is handled in both `pyproject.toml` and `package.json`.

Here are some guidelines (recommendations) for contributing to the `aioswitcher` project: * Code docstrings documentation [here](<https://aioswitcher.readthedocs.io/en/stable/codedocs.html>)

- For any change in dependencies, please use `pyscripts/poetry-to-requirements.py` for creating a valid `requirements.txt` file and add it to your PR. This is also done automatically with the `py37 testenv` in `tox`.
- While not all the test steps in `CircleCi` and in `Tox` are parallel to each other, most of them are, so tests failing with `Tox` will probably also fail with `CircleCi`.
- If writing *Python* code, please remember to [static type](<https://www.python.org/dev/peps/pep-0484/>).
- You can run `npm`'s script `spell-md-interactive` for handling all spelling mistakes before testing. You can also choose to run `spell-md-report` to print a full report instead of handling the spelling mistakes one-by-one. * `markdown-spellcheck` dictionary is the file `.spelling`.

10.6.1 NPM Scripts

Before using the scrips, you need to install the dependencies. From the `package.json` file path, run `npm install`, Then you can execute the scripts from the same path. * `npm run lint-md` will run `remark` against *markdown* files.

- `npm run validate-pkg` will run `package-json-validator` against the `package.json` file.
- `npm run spell-md-interactive` will run `markdown-spellcheck` against *markdown* files in an interactive manner allowing us to select the appropriate action.
- `npm run spell-md-report` will run `markdown-spellcheck` against *markdown* files and print the report to `stdout`.

10.7 Code of Conduct

The code of conduct can be found [here](<https://aioswitcher.readthedocs.io/en/stable/conduct.html>).

CODE DOCUMENTATION

11.1 aioswitcher

Switcher water heater unofficial API and bridge.

11.2 aioswitcher.api

Switcher water heater unofficial API and bridge, API Object.

class `aioswitcher.api.SwitcherV2Api` (*loop: asyncio.events.AbstractEventLoop, ip_addr: str, phone_id: str, device_id: str, device_password: str*)

Representation of the SwitcherV2 API object.

Parameters

- **loop** – the event loop for the factory to run in.
- **ip_addr** – the ip address assigned to the device
- **phone_id** – the phone id of the desired device.
- **device_id** – the id of the desired device.
- **device_password** – the password of the desired device.

Todo:

- `control_device` takes a timer value that must be converted to hex before calling this method using the designated tool. On the other hand, the rest of the action methods' takes their arguments raw and use the appropriate tool for converting themselves. This is confusing and needs to be adjusted.
-

async `__aenter__()`

Enter SwitcherV2Api asynchronous context manager.

Returns This instance of `aioswitcher.api.SwitcherV2Api` as an awaitable.

async `__await__()`

Return SwitcherV2Api awaitable object.

Returns This instance of `aioswitcher.api.SwitcherV2Api`.

async `__aexit__(exc_type, exc_value, traceback)`

Exit SwitcherV2Api asynchronous context manager.

async connect ()

Connect to asynchronous socket and get reader and writer object.

async disconnect ()

Disconnect from asynchronous socket.

async _full_login ()

Use for sending the login packet to the device.

Returns

A tuple of two `Tuple[str, Optional[messages.SwitcherV2LoginResponseMSG]]`.

The first object is a string containing the hexadecimal representation of the current unix timestamp.

The second object will be An instance of the serialized object `aioswitcher.api.messages.SwitcherV2LoginResponseMSG`.

Note: This is a private function, please consider using its wrapper function `login` instead.

async login ()

Use as wrapper for sending the login packet to the device.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2LoginResponseMSG`.

async _full_get_state ()

Use for sending the get state packet to the device.

Returns

A tuple of three `Tuple[str, Optional[messages.SwitcherV2LoginResponseMSG], Optional[messages.SwitcherV2StateResponseMSG]]`.

The first object is a string containing the hexadecimal representation of the current unix timestamp.

The second object will be An instance of the serialized object `aioswitcher.api.messages.SwitcherV2LoginResponseMSG`.

The third object will be An instance of the serialized object `aioswitcher.api.messages.SwitcherV2StateResponseMSG`.

Note: This is a private function, please consider using its wrapper function `get_state` instead.

async get_state ()

Use as wrapper for sending the get state packet to the device.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2StateResponseMSG`.

async set_auto_shutdown (full_time)

Use for sending the set auto-off packet to the device.

Parameters full_time – timedelta value containing the configuration value for auto-shutdown. Accepts anything between 1 and 3 hours.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG`.

async set_device_name (*name*)

Use for sending the set name packet to the device.

Parameters **name** – string name with the length of $2 \leq x \leq 32$.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG`.

async get_schedules ()

Use for retrieval of the schedules from the device.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG`.

async disable_enable_schedule (*schedule_data*)

Use for disabling or enabling a schedule on the device.

Parameters **schedule_data** – formatted data for updating the schedule, can be obtained from the `aioswitcher.schedules.SwitcherV2Schedule` object or created using the format `aioswitcher.consts.SCHEDULE_CREATE_DATA_FORMAT` filled with three values: weekdays, start-time and end-time. Weekdays can be created using the tool `aioswitcher.tools.create_weekdays_value`, the start and end times can be created using the `aioswitcher.tools.timedelta_str_to_schedule_time`

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG`.

async delete_schedule (*schedule_id*)

Use for deleting a schedule from the device.

Parameters **schedule_id** – the id of the schedule slot from the device, can be 0-7 as there are 8 slots available. Can be obtained from the `aioswitcher.schedules.SwitcherV2Schedule` object.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG`.

async create_schedule (*schedule_data*)

Use for creating a new schedule in the next empty schedule slot.

Parameters **schedule_data** – formatted data for updating the schedule, can be created using the format `aioswitcher.consts.SCHEDULE_CREATE_DATA_FORMAT` filled with three values: weekdays, start-time and end-time. Weekdays can be created using the tool `aioswitcher.tools.create_weekdays_value`, the start and end times can be created using the `aioswitcher.tools.timedelta_str_to_schedule_time`

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG`.

property connected

Return true if api is connected.

Type bool

11.3 aioswitcher.api.messages

Switcher Packet Response Messages.

class aioswitcher.api.messages.**ResponseMessageType**

Bases: enum.Enum

An enumeration.

class aioswitcher.api.messages.**SwitcherV2BaseResponseMSG** (*loop:* *asyncio.events.AbstractEventLoop*,
response: *bytes*,
msg_type: *aioswitcher.api.messages.ResponseMessageType*)

Representation of the switcher v2 base response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.
- **msg_type** – the message type as described in the “ResponseMessageType” Enum class.

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful

Indicating whether or not the request was successful.

Type bool

property unparsed_response

Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2ControlResponseMSG** (*loop:* *asyncio.events.AbstractEventLoop*,
response: *bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 control response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful

Indicating whether or not the request was successful.

Type bool

property unparsed_response

Return The raw response from the device.

Type bytes

```
class aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG (loop:
                                                                    async-
                                                                    cio.events.AbstractEventLoop,
                                                                    response:
                                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 create schedule response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

```
class aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG (loop:
                                                                    async-
                                                                    cio.events.AbstractEventLoop,
                                                                    response:
                                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 delete schedule response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

```
class aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG (loop:
                                                                    async-
                                                                    cio.events.AbstractEvent
                                                                    re-
                                                                    sponse:
                                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 dis/en schedule response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful

Indicating whether or not the request was successful.

Type bool

property unparsed_response

Return The raw response from the device.

Type bytes

```
class aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG (loop: asyncio.events.AbstractEventLoop,  
response: bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

representation of the switcher v2 get schedule message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

Todo:

- the `get_schedules` attribute should be a method.
 - `schedule_details` is `__init__` is yielding `List[str]` instead of `List[bytes]`, that's not supposed to happen.
-

property found_schedules

Return true if found schedules in the response.

Type bool

property get_schedules

Return schedules.

Type list(*aioswitcher.schedules.SwitcherV2Schedule*)

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful

Indicating whether or not the request was successful.

Type bool

property unparsed_response

Return The raw response from the device.

Type bytes

```
class aioswitcher.api.messages.SwitcherV2LoginResponseMSG (loop:          asyn-
                                                    cio.events.AbstractEventLoop,
                                                    response: bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 login response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType***property session_id**

Return the retrieved session id.

Type str**property successful**

Indicating whether or not the request was successful.

Type bool**property unparsed_response**

Return The raw response from the device.

Type bytes

```
class aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG (loop:          asyn-
                                                                    cio.events.AbstractEventLoop,
                                                                    response:
                                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 set auto-off response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType***property successful**

Indicating whether or not the request was successful.

Type bool**property unparsed_response**

Return The raw response from the device.

Type bytes

```
class aioswitcher.api.messages.SwitcherV2StateResponseMSG (loop: async-  
 cio.events.AbstractEventLoop,  
 response: bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 state response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

Todo:

- replace `init_future` attribute with `get_init_future` method.
-

async initialize (*response*)

Finish the initialization of the message and update the future object.

Parameters **response** – the raw response from the device.

property auto_off

Return the auto-off configuration value.

Type str

property current

Return the power consumption in amps.

Type float

property init_future

Return the future of the initialization.

Type asyncio.Future

async initialize (*response: bytes*) → None

Finish the initialization of the message and update the future object.

Parameters **response** – the raw response from the device.

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property power

Return the current power consumption in watts.

Type int

property state

Return the state.

Type str

property successful

Indicating whether or not the request was successful.

Type bool

property time_left

Return the time left to auto-off.

Type str

property unparsed_response

Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2UpdateNameResponseMSG** (*loop: asyncio.events.AbstractEventLoop, response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation of the switcher v2 update name response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type

the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful

Indicating whether or not the request was successful.

Type bool

property unparsed_response

Return The raw response from the device.

Type bytes

11.4 aioswitcher.api.packets

Switcher water heater unofficial API and bridge, API Packet formats.

11.5 aioswitcher.bridge

Switcher water heater unofficial API and bridge, Bridge Object.

class aioswitcher.bridge.**SwitcherV2Bridge** (*loop: asyncio.events.AbstractEventLoop, phone_id: str, device_id: str, device_password: str*)

Representation of the SwitcherV2 Bridge object.

Parameters

- **loop** – the event loop for the factory to run in.
- **phone_id** – the phone id of the desired device.
- **device_id** – the id of the desired device.
- **device_password** – the password of the desired device.

Todo:

- replace `queue` attribute with `get_queue` method.
-

async `__aenter__()`

Enter `SwitcherV2Bridge` asynchronous context manager.

Returns This instance of `aioswitcher.bridge.SwitcherV2Bridge` as an awaitable.

async `__await__()`

Return `SwitcherV2Bridge` awaitable object.

Returns This instance of `aioswitcher.bridge.SwitcherV2Bridge`.

async `__aexit__(exc_type, exc_value, traceback)`

Exit the `SwitcherV2Bridge` asynchronous context manager.

async `start()`

Create an asynchronous listener and start the bridge event.

async `stop()`

Stop the asynchronous bridge.

property `queue`

Return the queue storing updated device objects.

Type `asyncio.Queue`

property `running`

Return true if bridge is running.

Type `bool`

11.6 aioswitcher.bridge.messages

Switcher Bridge Response Messages.

```
class aioswitcher.bridge.messages.SwitcherV2BroadcastMSG (loop: asyncio.events.AbstractEventLoop,  
message: Union[bytes, str])
```

Representation of the `SwitcherV2` broadcast message.

Parameters

- **loop** – the event loop to perform actions in.
- **message** – the raw message from the device.

Todo:

- replace `init_future` attribute with `get_init_future` method.
-

async `initialize(message)`

Finish the initialization and update the future object.

Parameters **message** – the raw message from the device.

property `auto_off_set`

Return the auto-off configuration value.

Type `str`

property current

Return the power consumption in amps.

Type float

property device_id

Return the device id.

Type str

property device_state

Return the state of the device.

Type str

property init_future

Return the future of the device initialization.

Type asyncio.Future

property ip_address

Return the ip address.

Type str

property mac_address

Return the mac address.

Type str

property name

Return the device name.

Type str

property power

Return the power consumption in watts.

Type int

property remaining_time_to_off

Return the time left to auto-off.

Type str

property verified

Return whether or not the message is a SwitcherV2 message.

Type bool

11.7 aioswitcher.consts

Switcher water heater unofficial API and bridge, Global constants.

11.8 aioswitcher.devices

Switcher water heater unofficial API and bridge, Device classes.

```
class aioswitcher.devices.SwitcherV2Device (device_id: str, ip_address: str, mac_address: str, name: str, state: str, remaining_time: Optional[str], auto_off_set: str, power_consumption: int, electric_current: float, phone_id: str, device_password: str, last_state_change: datetime.datetime)
```

Representation of the switcherv2 device.

Parameters

- **device_id** – the id retrieved from the device.
- **ip_address** – the ip address assigned to the device.
- **mac_address** – the mac address assigned to the device.
- **name** – the name of the device.
- **state** – the current state of the device (in/off).
- **remaining_time** – remaining time (if on).
- **auto_off_set** – configured value for auto shutdown.
- **power_consumption** – the current power consumption in watts.
- **electric_current** – the current power consumption in amps.
- **phone_id** – the phone id retrieved from the device.
- **device_password** – the password retrieved from the device.
- **last_state_change** – datetime of the last state change.

```
update_device_data (ip_address, name, state, remaining_time, auto_off_set, power_consumption, electric_current, last_state_change)
```

Update the device state and data.

Parameters

- **ip_address** – the ip address assigned to the device.
- **name** – the name of the device.
- **state** – the current state of the device (on/off).
- **remaining_time** – remaining time (if on).
- **auto_off_set** – configured value for auto shutdown.
- **power_consumption** – the current power consumption in watts.
- **electric_current** – the current power consumption in amps.
- **last_state_change** – datetime of the last state change.

```
as_dict ()
```

Return as dict.

Returns A dictionary representation of the object properties. Used to make the object json serializable.

```
property auto_off_set
```

Returns the auto-off configuration value.

Type str

property device_id

Returns the device id.

Type str

property device_password

Returns the device password.

Type str

property electric_current

returns the power consumption in amps.

Type float

property ip_addr

Returns the ip address.

Type str

property last_data_update

Returns timestamp of the last update.

Type datetime

property last_state_change

Returns timestamp of the last state change.

Type datetime

property mac_addr

Returns the mac address.

Type str

property name

Returns the device name.

Type str

property phone_id

Returns the phone id.

Type str

property power_consumption

Returns the power consumption in watts.

Type int

property remaining_time

Returns the time left to auto shutdown.

Type str

property state

Returns the device state.

Type str

11.9 aioswitcher.errors

Switcher water heater unofficial API and bridge, Exception classes.

exception aioswitcher.errors.**CalculationError**

Bases: Exception

Exception to be raised when cpu bound calculation is failing.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception aioswitcher.errors.**DecodingError**

Bases: Exception

Exception to be raised when cpu bound decoding is failing.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception aioswitcher.errors.**EncodingError**

Bases: Exception

Exception to be raised when cpu bound encoding is failing.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

11.10 aioswitcher.protocols

Switcher water heater unofficial API and bridge, Network protocols.

```
class aioswitcher.protocols.SwitcherV2UdpProtocolFactory (loop: asyncio.events.AbstractEventLoop,  
phone_id: str, device_id: str, device_password: str, queue: asyncio.queues.Queue,  
run_factory_evt: asyncio.locks.Event)
```

Bases: asyncio.protocols.DatagramProtocol

Representation of the Asyncio UDP protocol factory.

Parameters

- **loop** – the event loop for the factory to run in.
- **phone_id** – the phone id of the desired device.
- **device_id** – the id of the desired device.
- **device_password** – the password of the desired device.
- **queue** – a `asyncio.Queue` for the factory to save messages in.
- **asyncio.Event for signaling the factory to run.** (*run_factory_evt*) –

Todo:

- replace `factory_future` attribute with `get_factory_future` method.
-

connection_made (*transport*)

Call on connection established.

datagram_received (*data*, *addr*)

Call on datagram received.

error_received (*exc*)

Call on exception received.

connection_lost (*exc*)

Call on connection lost.

close_transport (*future*)

Call for closing the transport.

async_handle_incoming_messages (*data*, *addr*)

Use for Handling incoming messages.

get_device_from_message (*ip_addr*, *future*)

Use for extracting the device from the broadcast message.

close_transport (*future*: *_asyncio.Future*) → None

Call for closing the transport.

connection_lost (*exc*: *Optional[Exception]*) → None

Call on connection lost.

connection_made (*transport*: *asyncio.transports.BaseTransport*) → None

Call on connection established.

datagram_received (*data*: *Union[bytes, str]*, *addr*: *Tuple*) → None

Call on datagram received.

error_received (*exc*: *Optional[Exception]*) → None

Call on exception received.

property_factory_future

Representing the initialization status.

Type *asyncio.Future*

get_device_from_message (*ip_addr*: *str*, *future*: *_asyncio.Future*) → None

Use for extracting the device from the broadcast message.

async_handle_incoming_messages (*data*: *Union[bytes, str]*, *addr*: *Tuple*) → None

Use for Handling incoming messages.

pause_writing ()

Called when the transport's buffer goes over the high-water mark.

Pause and resume calls are paired – `pause_writing()` is called once when the buffer goes strictly over the high-water mark (even if subsequent writes increases the buffer size even more), and eventually `resume_writing()` is called once when the buffer size reaches the low-water mark.

Note that if the buffer size equals the high-water mark, `pause_writing()` is not called – it must go strictly over. Conversely, `resume_writing()` is called when the buffer size is equal or lower than the low-water mark. These end conditions are important to ensure that things go as expected when either mark is zero.

NOTE: This is the only Protocol callback that is not called through `EventLoop.call_soon()` – if it were, it would have no effect when it's most needed (when the app keeps writing without yielding until `pause_writing()` is called).

resume_writing ()

Called when the transport's buffer drains below the low-water mark.

See `pause_writing()` for details.

11.11 aioswitcher.schedules

Switcher water heater unofficial API and bridge, Schedules.

```
class aioswitcher.schedules.SwitcherV2Schedule (loop: async-  
 cio.events.AbstractEventLoop, idx:  
 int, schedule_details: List[bytes])
```

Representation of the SwitcherV2 schedule slot.

Parameters

- **loop** – the event loop to perform schedule operation in.
- **idx** – the index of the schedule slot (0-7).
- **schedule_details** – the string raw schedule data details.

Todo:

- Replace `init_future` attribute with `get_init_future` method.
-

async initialize (*idx, schedule_details*)
Finish the initialization of the schedule.

as_dict ()
Return as dict.

Returns A dictionary representation of the object properties. Used to make the object json serializable.

property days
Return the weekdays of the schedule.

Type list(str)

property duration
Return the duration of the schedule.

Type str

property enabled
Return true if enabled, setter included.

Type bool

property end_time
Return the end time of the schedule.

Type str

property init_future
Return the future of the initialization.

Type asyncio.Future

property recurring
Return true if recurring.

Type bool

property schedule_data
Return the schedule data, setter included.

Type bytes

property `schedule_id`

Return the schedule id.

Type str

property `start_time`

Return the start time of the schedule.

Type str

`schedules._calc_next_run_for_schedule` (*schedule_details*)

Calculate the next runtime of the schedule.

Parameters `schedule_details` – `SwitcherV2Schedule` representing the schedule slot.

Returns A pretty string describing the next due run. e.g. “Due tomorrow at 17:00”.

Note: This is a private function containing blocking code. Please consider using `calc_next_run_for_schedule` (without the `_`), to schedule as a task on the event loop.

async `schedules.calc_next_run_for_schedule` (*loop*, *schedule_details*)

Asynchronous wrapper for `_calc_next_run_for_schedule`.

Use as async wrapper for calling `_calc_next_run_for_schedule`, calculating the next runtime of the schedule.

Parameters

- `loop` – the event loop to execute the function in.
- `schedule_details` – `SwitcherV2Schedule` representing the schedule slot.

Returns A pretty string describing the next due run. e.g. “Due tomorrow at 17:00”.

11.12 aioswitcher.tools

Switcher water heater unofficial API and bridge, Tools and Helpers.

`aioswitcher.tools._convert_minutes_to_timer` (*minutes: str*) → str

Convert on-timer minutes to hexadecimal before sending.

Parameters `minutes` – on-timer minutes to convert.

Returns Hexadecimal representation of the minutes argument.

Raises `aioswitcher.errors.EncodingError` – when failed hexlify.

Note: This is a private function containing blocking code. Please consider using `convert_minutes_to_timer` (without the `_`), to schedule as a task in the event loop.

`aioswitcher.tools._convert_seconds_to_iso_time` (*all_seconds: int*) → str

Convert seconds to iso time.

Parameters `all_seconds` – the total number of seconds to convert.

Returns `%M:%S` format. e.g. “02:24:37”.

Return type A string representing the converted iso time in `%H`

Raises `aioswitcher.errors.CalculationError` – when failed to convert the argument.

Note: This is a private function containing blocking code. Please consider using `convert_seconds_to_iso_time` (without the `_`), to schedule as a task in the event loop.

`aioswitcher.tools._convert_string_to_device_name` (*name: str*) → str
Convert string device name to hexadecimal before sending.

Parameters `name` – the desired name for conerting.

Returns Hexadecimal representation of the name argument.

Raises `aioswitcher.errors.EncodingError` – when failed hexlify.

Note: This is a private function containing blocking code. Please consider using `convert_string_to_device_name` (without the `_`), to schedule as a task in the event loop.

`aioswitcher.tools._convert_timedelta_to_auto_off` (*full_time: datetime.timedelta*) → str
Convert timedelta object for auto-shutdown to hexadecimal.

Parameters `full_time` – timedelta object representing the auto-shutdown time.

Returns Hexadecimal representation of the full_time argument.

Raises `aioswitcher.errors.EncodingError` – when failed hexlify.

Note: This is a private function containing blocking code. Please consider using `convert_timedelta_to_auto_off` (without the `_`), to schedule as a task in the event loop.

`aioswitcher.tools._crc_sign_full_packet_com_key` (*data: str*) → str
Calculate the crc for packets before send.

Parameters `data` – packet data to sign.

Returns The calculated and signed packet data.

Raises `aioswitcher.errors.EncodingError` – when failed to sign the packet.

Note: This is a private function containing blocking code. Please consider using `crc_sign_full_packet_com_key` (without the `_`), to schedule as a task in the event loop.

`aioswitcher.tools._create_weekdays_value` (*requested_days: List[int]*) → str
Create hex value from list of requested days for schedule updating.

Parameters `data` – list of integers representing the requested days. check `aioswitcher.consts` for the correct values.

Returns Hexadecimal representation of the days list.

Raises `aioswitcher.errors.EncodingError` – when failed to convert the argument.

Note: This is a private function containing blocking code. Please consider using `create_weekdays_value` (without the `_`), to schedule as a task in the event loop.

```
aioswitcher.tools._get_days_list_from_bytes (data: int) → List[str]
```

Extract week days from shchedule bytes data.

Parameters `data` – bytes representing the days list.

Returns List of string representation the week days included in the list. See `aioswitcher.consts` for days literals.

Raises `aioswitcher.errors.DecodingError` – when failed to analyze the argument.

Note: This is a private function containing blocking code. Please consider using `get_days_list_from_bytes` (without the `_`), to schedule as a task in the event loop.

```
aioswitcher.tools._get_time_from_bytes (data: bytes) → str
```

Extract start/end time from schedule bytes.

Parameters `data` – bytes representing the start or the end time for the schedule.

Returns `%M` format. e.g. “20:30”.

Return type Time string in `%H`

Raises `aioswitcher.errors.DecodingError` – when failed to analyze the argument.

Note: This is a private function containing blocking code. Please consider using `get_time_from_bytes` (without the `_`), to schedule as a task in the event loop.

```
aioswitcher.tools._get_timestamp () → str
```

Generate hexadecimal representation of the current timestamp.

Returns Hexadecimal representation of the current unix time retrieved by `time.time`.

Raises `aioswitcher.errors.DecodingError` – when failed to analyze the timestamp.

Note: This is a private function containing blocking code. Please consider using `get_timestamp` (without the `_`), to schedule as a task in the event loop.

```
aioswitcher.tools._timedelta_str_to_schedule_time (time_value: str) → str
```

Convert time string to schedule start/end time to hexadecimal.

Parameters `data` – time to convert. e.g. “21:00”.

Returns Hexadecimal representation of `time_value` argument.

Raises `aioswitcher.errors.EncodingError` – when failed to convert the argument.

Note: This is a private function containing blocking code. Please consider using `timedelta_str_to_schedule_time` (without the `_`), to schedule as a task in the event loop.

```
async aioswitcher.tools.convert_minutes_to_timer (loop: aio.events.AbstractEventLoop,
minutes: str) → str
```

Asynchronous wrapper for `_convert_minutes_to_timer`.

Use as `async` wrapper for calling `_convert_minutes_to_timer`, converting on-timer minutes to hexadecimal.

Parameters `minutes` – on-timer minutes to convert.

Returns Hexadecimal representation of the mintues argument.

async `aioswitcher.tools.convert_seconds_to_iso_time` (*loop*: `asyncio.events.AbstractEventLoop`,
all_seconds: `int`) → `str`

Asynchronous wrapper for `_convert_seconds_to_iso_time`.

Use as async wrapper for calling `_convert_seconds_to_iso_time`, calculating the next runtime of the schedule.

Parameters

- **loop** – the event loop to execute the function in.
- **all_seconds** – the total number of seconds to convert.

Returns `%M:%S` format. e.g. “02:24:37”.

Return type A string represnting the converted iso time in `%H`

async `aioswitcher.tools.convert_string_to_device_name` (*loop*: `asyncio.events.AbstractEventLoop`,
name: `str`) → `str`

Asynchronous wrapper for `_convert_string_to_device_name`.

Use as async wrapper for calling `_convert_string_to_device_name`, converting name to hexadecimal.

Parameters **name** – the desired name for conerting.

Returns Hexadecimal representation of the name argument.

async `aioswitcher.tools.convert_timedelta_to_auto_off` (*loop*: `asyncio.events.AbstractEventLoop`,
full_time: `datetime.timedelta`) → `str`

Asynchronous wrapper for `_convert_timedelta_to_auto_off`.

Use as async wrapper for calling `_convert_timedelta_to_auto_off`, converting timedelta auto-shutdown configuration to hexadecimal.

Parameters **full_time** – timedelta object represnting the auto-shutdown time.

Returns Hexadecimal representation of the full_time argument.

async `aioswitcher.tools.crc_sign_full_packet_com_key` (*loop*: `asyncio.events.AbstractEventLoop`,
data: `str`) → `str`

Asynchronous wrapper for `_crc_sign_full_packet_com_key`.

Use as async wrapper for calling `_crc_sign_full_packet_com_key`, performing crc sign to the packet data.

Parameters **data** – packet data to sign.

Returns The calculated and signed packet data.

async `aioswitcher.tools.create_weekdays_value` (*loop*: `asyncio.events.AbstractEventLoop`,
requested_days: `List[int]`) → `str`

Asynchronous wrapper for `_create_weekdays_value`.

Use as async wrapper for calling `_create_weekdays_value`, creating hex value from list of requested days for schedule updating.

Parameters **data** – list of integers represnting the requested days. check `aioswitcher.consts` for the correct values.

Returns Hexadecimal representation of the days list.

async aioswitcher.tools.get_days_list_from_bytes (loop: *asyncio.events.AbstractEventLoop*, data: *int*) → List[str]

Asynchronous wrapper for `_get_days_list_from_bytes`.

Use as async wrapper for calling `_get_days_list_from_bytes`, extracting week days from schedule bytes data.

Parameters `data` – bytes representing the days list.

Returns List of string representation the week days included in the list. See `aioswitcher.consts` for days literals.

async aioswitcher.tools.get_time_from_bytes (loop: *asyncio.events.AbstractEventLoop*, data: *bytes*) → str

Asynchronous wrapper for `_get_time_from_bytes`.

Use as async wrapper for calling `_get_time_from_bytes`, extracting start/end time from schedule bytes data.

Parameters `data` – bytes representing the start or the end time for the schedule.

Returns %M format. e.g. “20:30”.

Return type Time string in %H

async aioswitcher.tools.get_timestamp (loop: *asyncio.events.AbstractEventLoop*) → str

Asynchronous wrapper for `_get_timestamp`.

Use as async wrapper for calling `_get_timestamp`, creating hexadecimal representation of the current timestamp.

Returns Hexadecimal representation of the current unix time retrieved by `time.time`.

async aioswitcher.tools.timedelta_str_to_schedule_time (loop: *asyncio.events.AbstractEventLoop*, time_value: *str*) → str

Asynchronous wrapper for `_timedelta_str_to_schedule_time`.

Use as async wrapper for calling `_timedelta_str_to_schedule_time`, converting time string to schedule start/end time to hexadecimal.

Parameters `data` – time to convert. e.g. “21:00”.

Returns Hexadecimal representation of `time_value` argument.

PYTHON MODULE INDEX

a

- aioswitcher, 33
- aioswitcher.api, 33
- aioswitcher.api.messages, 35
- aioswitcher.api.packets, 41
- aioswitcher.bridge, 41
- aioswitcher.bridge.messages, 42
- aioswitcher.consts, 43
- aioswitcher.devices, 43
- aioswitcher.errors, 45
- aioswitcher.protocols, 46
- aioswitcher.schedules, 48
- aioswitcher.tools, 49

Symbols

`__aenter__()` (*aioswitcher.api.SwitcherV2Api* method), 33
`__aenter__()` (*aioswitcher.bridge.SwitcherV2Bridge* method), 42
`__aexit__()` (*aioswitcher.api.SwitcherV2Api* method), 33
`__aexit__()` (*aioswitcher.bridge.SwitcherV2Bridge* method), 42
`__await__()` (*aioswitcher.api.SwitcherV2Api* method), 33
`__await__()` (*aioswitcher.bridge.SwitcherV2Bridge* method), 42
`_calc_next_run_for_schedule()` (*aioswitcher.schedules* method), 49
`_convert_minutes_to_timer()` (in module *aioswitcher.tools*), 49
`_convert_seconds_to_iso_time()` (in module *aioswitcher.tools*), 49
`_convert_string_to_device_name()` (in module *aioswitcher.tools*), 50
`_convert_timedelta_to_auto_off()` (in module *aioswitcher.tools*), 50
`_crc_sign_full_packet_com_key()` (in module *aioswitcher.tools*), 50
`_create_weekdays_value()` (in module *aioswitcher.tools*), 50
`_full_get_state()` (*aioswitcher.api.SwitcherV2Api* method), 34
`_full_login()` (*aioswitcher.api.SwitcherV2Api* method), 34
`_get_days_list_from_bytes()` (in module *aioswitcher.tools*), 50
`_get_time_from_bytes()` (in module *aioswitcher.tools*), 51
`_get_timestamp()` (in module *aioswitcher.tools*), 51
`_timedelta_str_to_schedule_time()` (in module *aioswitcher.tools*), 51

A

aioswitcher (module), 33

aioswitcher.api (module), 33
aioswitcher.api.messages (module), 35
aioswitcher.api.packets (module), 41
aioswitcher.bridge (module), 41
aioswitcher.bridge.messages (module), 42
aioswitcher.consts (module), 43
aioswitcher.devices (module), 43
aioswitcher.errors (module), 45
aioswitcher.protocols (module), 46
aioswitcher.schedules (module), 48
aioswitcher.tools (module), 49
`as_dict()` (*aioswitcher.devices.SwitcherV2Device* method), 44
`as_dict()` (*aioswitcher.schedules.SwitcherV2Schedule* method), 48
`auto_off()` (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 40
`auto_off_set()` (*aioswitcher.bridge.messages.SwitcherV2BroadcastM* property), 42
`auto_off_set()` (*aioswitcher.devices.SwitcherV2Device* property), 44

C

`calc_next_run_for_schedule()` (*aioswitcher.schedules* method), 49
CalculationError, 45
`close_transport()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 47
`connect()` (*aioswitcher.api.SwitcherV2Api* method), 33
`connected()` (*aioswitcher.api.SwitcherV2Api* property), 35
`connection_lost()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 47
`connection_made()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 46, 47
`convert_minutes_to_timer()` (in module *aioswitcher.tools*), 51

`convert_seconds_to_iso_time()` (in module `aioswitcher.tools`), 52
`convert_string_to_device_name()` (in module `aioswitcher.tools`), 52
`convert_timedelta_to_auto_off()` (in module `aioswitcher.tools`), 52
`crc_sign_full_packet_com_key()` (in module `aioswitcher.tools`), 52
`create_schedule()` (`aioswitcher.api.SwitcherV2Api` method), 35
`create_weekdays_value()` (in module `aioswitcher.tools`), 52
`current()` (`aioswitcher.api.messages.SwitcherV2StateResponseMSG` property), 40
`current()` (`aioswitcher.bridge.messages.SwitcherV2BroadcastMSG` property), 42
D
`datagram_received()` (`aioswitcher.protocols.SwitcherV2UdpProtocolFactory` method), 47
`days()` (`aioswitcher.schedules.SwitcherV2Schedule` property), 48
`DecodingError`, 46
`delete_schedule()` (`aioswitcher.api.SwitcherV2Api` method), 35
`device_id()` (`aioswitcher.bridge.messages.SwitcherV2BroadcastMSG` property), 43
`device_id()` (`aioswitcher.devices.SwitcherV2Device` property), 44
`device_password()` (`aioswitcher.devices.SwitcherV2Device` property), 45
`device_state()` (`aioswitcher.bridge.messages.SwitcherV2BroadcastMSG` property), 43
`disable_enable_schedule()` (`aioswitcher.api.SwitcherV2Api` method), 35
`disconnect()` (`aioswitcher.api.SwitcherV2Api` method), 34
`duration()` (`aioswitcher.schedules.SwitcherV2Schedule` property), 48
E
`electric_current()` (`aioswitcher.devices.SwitcherV2Device` property), 45
`enabled()` (`aioswitcher.schedules.SwitcherV2Schedule` property), 48
`EncodingError`, 46
`end_time()` (`aioswitcher.schedules.SwitcherV2Schedule` property), 48
`error_received()` (`aioswitcher.protocols.SwitcherV2UdpProtocolFactory` method), 47
F
`factory_future()` (`aioswitcher.protocols.SwitcherV2UdpProtocolFactory` property), 47
`found_schedules()` (`aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG` property), 38
G
`get_days_list_from_bytes()` (in module `aioswitcher.tools`), 52
`get_device_from_message()` (`aioswitcher.protocols.SwitcherV2UdpProtocolFactory` method), 47
`get_schedules()` (`aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG` property), 38
`get_schedules()` (`aioswitcher.api.SwitcherV2Api` method), 35
`get_state()` (`aioswitcher.api.SwitcherV2Api` method), 34
`get_time_from_bytes()` (in module `aioswitcher.tools`), 53
`get_timestamp()` (in module `aioswitcher.tools`), 53
H
`handle_incoming_messages()` (`aioswitcher.protocols.SwitcherV2UdpProtocolFactory` method), 47
I
`init_future()` (`aioswitcher.api.messages.SwitcherV2StateResponseMSG` property), 40
`init_future()` (`aioswitcher.bridge.messages.SwitcherV2BroadcastMSG` property), 43
`init_future()` (`aioswitcher.schedules.SwitcherV2Schedule` property), 48
`initialize()` (`aioswitcher.api.messages.SwitcherV2StateResponseMSG` method), 40
`initialize()` (`aioswitcher.bridge.messages.SwitcherV2BroadcastMSG` method), 42
`initialize()` (`aioswitcher.schedules.SwitcherV2Schedule` method), 48
`ip_addr()` (`aioswitcher.devices.SwitcherV2Device` property), 45
`ip_address()` (`aioswitcher.bridge.messages.SwitcherV2BroadcastMSG` property), 43
L
`last_data_update()` (`aioswitcher.devices.SwitcherV2Device` property), 45

last_state_change() (*aioswitcher.devices.SwitcherV2Device* property), 45
 login() (*aioswitcher.api.SwitcherV2Api* method), 34

M

mac_addr() (*aioswitcher.devices.SwitcherV2Device* property), 45
 mac_address() (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 43
 msg_type() (*aioswitcher.api.messages.SwitcherV2BaseResponseMSG* property), 36
 msg_type() (*aioswitcher.api.messages.SwitcherV2ControlResponseMSG* property), 36
 msg_type() (*aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG* property), 37
 msg_type() (*aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG* property), 37
 msg_type() (*aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG* property), 38
 msg_type() (*aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG* property), 38
 msg_type() (*aioswitcher.api.messages.SwitcherV2LoginResponseMSG* property), 39
 msg_type() (*aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG* property), 39
 msg_type() (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 40
 msg_type() (*aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG* property), 41

N

name() (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 43
 name() (*aioswitcher.devices.SwitcherV2Device* property), 45

P

pause_writing() (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 47
 phone_id() (*aioswitcher.devices.SwitcherV2Device* property), 45
 power() (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 40
 power() (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 43
 power_consumption() (*aioswitcher.devices.SwitcherV2Device* property), 45

Q

queue() (*aioswitcher.bridge.SwitcherV2Bridge* property), 42

R

recurring() (*aioswitcher.schedules.SwitcherV2Schedule* property), 48
 remaining_time() (*aioswitcher.devices.SwitcherV2Device* property), 45
 remaining_time_to_off() (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 43
 response_message_type() (*aioswitcher.api.messages*), 35
 resume_writing() (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 47
 response() (*aioswitcher.bridge.SwitcherV2Bridge* property), 42
 schedule_id() (*aioswitcher.schedules.SwitcherV2Schedule* property), 48
 session_id() (*aioswitcher.api.messages.SwitcherV2LoginResponseMSG* property), 39
 set_auto_shutdown() (*aioswitcher.api.SwitcherV2Api* method), 35
 set_device_name() (*aioswitcher.api.SwitcherV2Api* method), 35
 start() (*aioswitcher.bridge.SwitcherV2Bridge* method), 42
 start_time() (*aioswitcher.schedules.SwitcherV2Schedule* property), 49
 state() (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 40
 state() (*aioswitcher.devices.SwitcherV2Device* property), 45
 stop() (*aioswitcher.bridge.SwitcherV2Bridge* method), 42
 success() (*aioswitcher.api.messages.SwitcherV2BaseResponseMSG* property), 36
 successful() (*aioswitcher.api.messages.SwitcherV2ControlResponseMSG* property), 36
 successful() (*aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG* property), 37
 successful() (*aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG* property), 37
 successful() (*aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG* property), 38
 successful() (*aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG* property), 38
 successful() (*aioswitcher.api.messages.SwitcherV2LoginResponseMSG* property), 39
 successful() (*aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG* property), 39

successful () (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 40

successful () (*aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG* property), 41

SwitcherV2Api (*class in aioswitcher.api*), 33

SwitcherV2BaseResponseMSG (*class in aioswitcher.api.messages*), 36

SwitcherV2Bridge (*class in aioswitcher.bridge*), 41

SwitcherV2BroadcastMSG (*class in aioswitcher.bridge.messages*), 42

SwitcherV2ControlResponseMSG (*class in aioswitcher.api.messages*), 36

SwitcherV2CreateScheduleResponseMSG (*class in aioswitcher.api.messages*), 36

SwitcherV2DeleteScheduleResponseMSG (*class in aioswitcher.api.messages*), 37

SwitcherV2Device (*class in aioswitcher.devices*), 43

SwitcherV2DisableEnableScheduleResponseMSG (*class in aioswitcher.api.messages*), 37

SwitcherV2GetScheduleResponseMSG (*class in aioswitcher.api.messages*), 38

SwitcherV2LoginResponseMSG (*class in aioswitcher.api.messages*), 39

SwitcherV2Schedule (*class in aioswitcher.schedules*), 48

SwitcherV2SetAutoOffResponseMSG (*class in aioswitcher.api.messages*), 39

SwitcherV2StateResponseMSG (*class in aioswitcher.api.messages*), 39

SwitcherV2UdpProtocolFactory (*class in aioswitcher.protocols*), 46

SwitcherV2UpdateNameResponseMSG (*class in aioswitcher.api.messages*), 41

unparsed_response () (*aioswitcher.api.messages.SwitcherV2BaseResponseMSG* property), 36

unparsed_response () (*aioswitcher.api.messages.SwitcherV2ControlResponseMSG* property), 36

unparsed_response () (*aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG* property), 37

unparsed_response () (*aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG* property), 37

unparsed_response () (*aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG* property), 38

unparsed_response () (*aioswitcher.api.messages.SwitcherV2LoginResponseMSG* property), 39

unparsed_response () (*aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG* property), 39

unparsed_response () (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 41

unparsed_response () (*aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG* property), 41

update_device_data () (*aioswitcher.devices.SwitcherV2Device* method), 44

verified () (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 43

V

W

with_traceback () (*aioswitcher.errors.CalculationError* method), 46

with_traceback () (*aioswitcher.errors.DecodingError* method), 46

with_traceback () (*aioswitcher.errors.EncodingError* method), 46

T

time_left () (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 40

timedelta_str_to_schedule_time () (*in module aioswitcher.tools*), 53

U

unparsed_response () (*aioswitcher.api.messages.SwitcherV2BaseResponseMSG* property), 36

unparsed_response () (*aioswitcher.api.messages.SwitcherV2ControlResponseMSG* property), 36

unparsed_response () (*aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG* property), 37

unparsed_response () (*aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG* property), 37